DETAILED ACTION

1. The Office Action is responsive to the communication filed on 08/06/2009.

2. Claims 1-20 are pending in the application.

Re-setting of Response Period

3. Applicant contacted the Examiner within one month of receipt of the non-final rejection. Applicant informed Examiner that the 1.132 affidavit was not clearly considered. On this basis, this office action is being issued.

Continued Examination Under 37 CFR 1.114

4. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on 08/06/2009 has been entered.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

6. The factual inquiries set forth in *Graham* **v.** *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

5. Claims 1-20 are rejected under 35 U.S.C. 103(a) as being unpatentable over Claims 1-20 are rejected under 35 U.S.C. 103(a) as being unpatentable over Coker et al. (USPN 2007/0250840) in view over Lee et al. (USPN 20040088700) and in further view over Balducci et al. (USPN 20040103174)

6. As per claims 1 and 10, Coker et al. teaches which code (i) is configured to be stored on the client device and be executed during each of subsequent communications between the client device and the server device ([0307-0309] e.g., client includes a component called a busy state manager configured to monitor and inform a user of a status and progress of the submitted request), and (ii) when executed blocks the client device from receiving user input during the communications between the client device and the server device ([0309] e.g., client can inform the user that request processing has started and lock the user interface), determines whether any of the communications between the client device and the server device lasts longer than a specific time, and, upon determining that the specific time has been exceeded, causes a message provided in the code to be presented to a user of the client device ([0309-0310] e.g., upon determining that the request from the client may take a long time to process, the server will

notify the client accordingly....the client can update the progress bar to show how much of the task has been completed at that point in time.

However, Coker et al. does not teach the server providing executable code to the client computer, i.e., providing the busy-state manager component to a client computer). Lee et al. teaches a system for automatically installing software on a client via a server ([ABSTRACT], [FIG 1])

Therefore, at the time the invention was made, one of ordinary skill in the art would have motivation to install client components using software stored on a server. Lee et al. teaches that enterprises employ client-server models to facilitate the configuration of a client computer via downloading necessary application software ([0004-0006]). Coker et al. teaches that various types of clients can be supported....the various types of clients including remote clients ([0063]), and in addition, teaches that clients can download a subset of server's data to use locally ([0070]). Therefore, in client-server interactions, as taught by Lee et al. it would have been obvious to enable a server to provide necessary components to remote clients to facilitate server-client interactions, including downloading necessary components to a client. Here, it would have been obvious to download a busy-state manager component to a remote client via an application server.

However, Coker et al., as modified, does not teach that the client device is configured to engage in communications with the server device for a plurality of application programs. Coker et al. does teach that the client computer makes requests to the server such that particular tasks may be executed on the server, and in response to the client request the client computer is locked ([0309-0310] e.g., it is interpreted that the client request is not tied to any particular program

such that any program on the client computer that should make a request to the server for server processing would result in the client computer becoming locked). Balducci et al. teaches an application program, i.e., Microsoft Outlook, on the client computer ([0024]) and in addition teaches that the client may delete content within the server ([0075]).

Therefore, it would have been obvious to one of ordinary skill in the art to apply the "lock mechanism" (e.g., client code used to lock the client computer) , as taught by Coker et al., to be applicable to any program (e.g., Outlook) on the client computer that would require external server processing. Since a client may send a delete request, for example, to a server that could take longer than a reasonable time (e.g., server is backed up), it would have been obvious to apply the "lock mechanism" to any program making such a request that would require the client to be locked. During the period in which the client is locked, the user would be informed that a request is taking longer than expected, as taught by Coker et al. Balducci et al. illustrates that a an application program, such as Outlook, may send a request to the server, in turn locking the client during this task, as taught by Coker et al. (e.g., as in the case of emptying a folder on the server or synchronizing a client and server)

7. As per claim 2, Coker et al. teaches the method of claim 1, wherein the executable code is client-side framework code provided from the framework code in the server device that controls communication between the server device and the client device ([0306-0310] e.g., busy state manager component is stored on the client, i.e. client side framework, provided from the framework code in the server device (e.g., as modified, an application server provides the busy state component to the client), where the framework code controls communication between the client and server, i.e., client is informed by the server communication will last longer than

expected, and in response, the client user interface is locked. Controlling communication is interpreted as corresponding to informing the client device of a process status)

8. As per claim 3, Lee et al. teaches the method of claim 1, further comprising providing the executable code in response to the server device receiving a request from the client device to launch at least one of the application program capable of initiating the communications ([0011] e.g., automatically installing software on a client via a client login request. As interpreted, a login request is an application program that initiates the request for the executable code. As applied to Coker et al., the busy-state component could be downloaded in response to the client request for the component by following the steps provided in paragraph 0011). Once the "lock mechanism" is installed in the client computer (e.g.., code may be installed on the client computer via a provisioning service), any program that could make a request to a server that requires a "lock" to be implemented during the request could be launched. For example, a user makes a delete request to the server using Outlook. The client is locked during the delete request. The client code would lock the client during the use Outlook delete request to the server. If the request takes longer than expected, the client informs the user.

9. As per claim 4, Coker et al., as modified, teaches the method of claim 3, further comprising providing application program code from at least on the application programs to the client device wherein the message is an over-definition of a default message that would otherwise be presented. (As modified by Coker et al., [0309], Outlook provides a status bar, i.e., code, during a delete request. The status bar is displayed to the user with a progress indicator, i.e., over-definition. As per the applicant's background section, "sudden displays of messages and sudden disappearances" are unnecessary. As applied to Coker et al., it would have been

obvious to display the indicator when the request is going to take longer than expected and to not display the progress bar as to avoid unnecessary disruptions, i.e., "otherwise be presented."

10. As per claim 5, Coker et al. teaches the method of claim 1, wherein a communication lasts longer than the specific time due to network delays, server-side delays, or combinations thereof ([0309] e.g., request ling-running server operations i.e., server-side delays)

11. As per claim 6, Coker et al. teaches the method of claim 1, wherein a communication lasts longer than the specific time when the client device has not displayed a server response within the specific time ([0309-0314] e.g., once the client is informed by the server that the request may take a long time to process in view of the requests from a client, it would have been obvious to provide an indication that the client request is taking longer than expected, i.e., not displaying a server response, to the client request)

12. As per claim 7, Coker et al. teaches the method of claim 1, wherein the executable code ceases to block the client device from receiving user input after each communication has ended ([0310] e.g., client continues to lock the interface until the request processing is completed)

13. As per claim 8, Coker et al. teaches the method of claim 1, wherein the executable code causes the message to be presented on the client device during one of the communication and causes the client device to cease presenting the message after that communication has ended ([0310] e.g., as best understood, the busy manager causes the notification to be presented to the user and causes the client to cease presenting the message when the request processing is completed)

14. As per claim 9, Coker et al. teaches the method of claim 1, further comprising setting the specific time based on at least one selected from the group consisting of: a roundtrip time for a

communication between the server device and the client device, typical roundtrip times for communication between the server device and the client device, a roundtrip time expected by at least one user of the client device, and combinations thereof ([0314] e.g., roundtrip to the server, including the request from the client to the server and a response notification to the server, in view of [0310] e.g., determining the request is taking longer than expected, is interpreted that determining a request may take a long time to process is a function of a round trip, i.e., request from a client to server and response notification)

15. As per claim 11, Coker et al., as modified, teaches a method of informing a user about communications between a client device and a server device, the method comprising:

storing the executable code on the client device, the executable code configured to be executed during each of subsequent communications between the client device and the server device ([0306-0310] e.g., busy-manager component);

blocking, per the executable code, the client device from receiving user input during its communications with a server device ([0309-0310]);

determining whether any of the communications lasts longer than a specific time; and presenting, per the executable code, a message provided in the code to a user of the client device upon determining that any of the communications lasts longer than the specific time ([0310] e.g. server notifies client that the request may take a long time to finish)

However, Coker et al. does not teach the server providing executable code to the client computer, i.e., providing the busy-state manager component to a client computer). Lee et al. teaches a system for automatically installing software on a client via a server ([ABSTRACT], [FIG 1])

Therefore, at the time the invention was made, one of ordinary skill in the art would have motivation to install client components using software stored on a server. Lee et al. teaches that enterprises employ client-server models to facilitate the configuration of a client computer via downloading necessary application software ([0004-0006]). Coker et al. teaches that various types of clients can be supported....the various types of clients including remote clients ([0063]), and in addition, teaches that clients can download a subset of server's data to use locally ([0070]). Therefore, in client-server interactions, as taught by Lee et al. it would have been obvious to enable a server to provide necessary components to remote clients to facilitate server-client interactions, including downloading necessary components to a client. Here, it would have been obvious to download a busy-state manager component to a remote client via an application server.

However, Coker et al., as modified, does not teach that the client device is configured to engage in communications with the server device for a plurality of application programs. Coker et al. does teach that the client computer makes requests to the server such that particular tasks may be executed on the server, and in response to the client request the client computer is locked ([0309-0310] e.g., it is interpreted that the client request is not tied to any particular program such that any program on the client computer that should make a request to the server for server processing would result in the client computer becoming locked). Balducci et al. teaches an application program, i.e., MicrosIft Outlook, on the client computer ([0024]) and in addition teaches that the client may delete content within the server ([0075]).

Therefore, it would have been obvious to one of ordinary skill in the art to apply the "lock mechanism" (e.g., client code used to lock the client computer) , as taught by Coker et al., to be

applicable to any program (e.g., Outlook) on the client computer that would require external server processing. Since a client may send a delete request, for example, to a server that could take longer than a reasonable time (e.g., server is backed up), it would have been obvious to apply the "lock mechanism" to any program making such a request that would require the client to be locked. During the period in which the client is locked, the user would be informed that a request is taking longer than expected, as taught by Coker et al. Balducci et al. illustrates that a an application program, such as Outlook, may send a request to the server, in turn locking the client during this task, as taught by Coker et al. (e.g., as in the case of emptying a folder on the server or synchronizing a client and server)

16. As per claim 12, Coker et al. teaches the method of claim 11, wherein the presented message is an over-definition of a default message e.g., supra claim 4 discussion)

17. As per claim 13, Coker et al. teaches the method of claim 11, further comprising setting the specific time based on at least one selected from the group consisting of: a roundtrip time for a communication between the server device and the client device, typical roundtrip times for communication between the server device and the client device, a roundtrip time expected by at least one user of the client device, and combinations thereof ([0314] e.g., roundtrip to the server, including the request from the client to the server and a response notification to the server, in view of [0310] e.g., determining the request is taking longer than expected, is interpreted that determining a request may take a long time to process is a function of a round trip, i.e., request from a client to server and response notification)

18. As per claim 14, Coker et al. teaches a computer program product containing executable instructions that when executed cause a processor to perform operations comprising:

block a client device from receiving user input during its communications with a server device ([0309] e.g., locking user interface);

determine whether any of the communications lasts longer than a specific time;

cause a message provided in the computer program product to be presented to a user of the client device if determining that any of the communications lasts longer than the specific time ([0309-0310]);

wherein the computer program product is configured to be provided from the server device to the client device, be stored on the client device and to be executed during each of the communications between the client device and the server device (e.g., supra claim 1)

19. As per claim 15, Coker et al. teaches a computer system comprising:

a server device with server-side framework code which when executed on the server device establishes a client-server framework for client-server communications ([Fig 2], [Fig 13] e.g., client-server communications with executable code),; and

code (i) is configured to be stored on the client device and be executed during each of subsequent communications between the client device and the server device ([0307-0309] e.g., client includes a component called a busy state manager configured to monitor and inform a user of a status and progress of the submitted request), and (ii) when executed blocks the client device from receiving user input during the communications between the client device and the server device ([0309] e.g., client can inform the user that request processing has started and lock the user interface), determines whether any of the communications between the client device and the server device lasts longer than a specific time, and, upon determining that the specific time has been exceeded, causes a message provided in the code to be presented to a user of the client

device ([0309-0310] e.g., upon determining that the request from the client may take a long time to process, the server will notify the client accordingly....the client can update the progress bar to show how much of the task has been completed at that point in time.

However, Coker et al. does not teach the server providing executable code to the client computer, i.e., providing the busy-state manager component to a client computer). Lee et al. teaches a system for automatically installing software on a client via a server ([ABSTRACT], [FIG 1])

Therefore, at the time the invention was made, one of ordinary skill in the art would have motivation to install client components using software stored on a server. Lee et al. teaches that enterprises employ client-server models to facilitate the configuration of a client computer via downloading necessary application software ([0004-0006]). Coker et al. teaches that various types of clients can be supported....the various types of clients including remote clients ([0063]), and in addition, teaches that clients can download a subset of server's data to use locally ([0070]). Therefore, in client-server interactions, as taught by Lee et al. it would have been obvious to enable a server to provide necessary components to remote clients to facilitate server-client interactions, including downloading necessary components to a client. Here, it would have been obvious to download a busy-state manager component to a remote client via an application server.

However, Coker et al., as modified, does not teach that the client device is configured to engage in communications with the server device for a plurality of application programs. Coker et al. does teach that the client computer makes requests to the server such that particular tasks may be executed on the server, and in response to the client request the client computer is locked

([0309-0310] e.g., it is interpreted that the client request is not tied to any particular program such that any program on the client computer that should make a request to the server for server processing would result in the client computer becoming locked). Balducci et al. teaches an application program, i.e., MicrosIft Outlook, on the client computer ([0024]) and in addition teaches that the client may delete content within the server ([0075]).

Therefore, it would have been obvious to one of ordinary skill in the art to apply the "lock mechanism" (e.g., client code used to lock the client computer) , as taught by Coker et al., to be applicable to any program (e.g., Outlook) on the client computer that would require external server processing. Since a client may send a delete request, for example, to a server that could take longer than a reasonable time (e.g., server is backed up), it would have been obvious to apply the "lock mechanism" to any program making such a request that would require the client to be locked. During the period in which the client is locked, the user would be informed that a request is taking longer than expected, as taught by Coker et al. Balducci et al. illustrates that a an application program, such as Outlook, may send a request to the server, in turn locking the client during this task, as taught by Coker et al. (e.g., as in the case of emptying a folder on the server or synchronizing a client and server)

20. As per claim 16, Coker et al. teaches the method of claim 15, wherein a communication lasts longer than the specific time due to network delays, server-side delays, or combinations thereof ([0309] e.g., request ling-running server operations i.e., server-side delays)

21. As per claim 17, Coker et al. teaches the method of claim 15, wherein the presented message is an over-definition of a default message (e.g., supra claim 4 discussion)

22. As per claim 18, Coker et al. teaches the computer system of claim 15, wherein the client-side framework code causes the message to be displayed on the client device ([0306-0310] e.g., busy state manager)

23. As per claim 19, Coker et al. teaches the computer system of claim 15, wherein the specific time is based on at least one selected from the group consisting of: typical roundtrip times for communication between the server device and the client device, a roundtrip time expected by at least one user of the client device, and combinations thereof ([0314] e.g., roundtrip to the server, including the request from the client to the server and a response notification to the server, in view of [0310] e.g., determining the request is taking longer than expected, is interpreted that determining a request may take a long time to process is a function of a round trip, i.e., request from a client to server and response notification)

24. As per claim 20, Coker et al. teaches the computer system of claim 15, wherein at least one roundtrip time for communication between the server device and the client device is recorded and the specific time is set based on the at least one roundtrip time ([0314] e.g., in light of 0309-0310, a determination is made that a request will take longer than expected. The roundtrip, i.e., request to client and notification from server to client, is understood as being part of the determination that a request received from a client may take a long time to process. Thus, the determination would require that the roundtrip time be known, and if this time (request by client and server response) is going to take longer than expected, a message would be presented to the user)

Response to Arguments

25. [A] Applicant argues that Coker et al. teaches that "the notification regarding the runtime delay is made from the server to the client. This differs fundamentally from the approach in the present subject matter, where a client device alerts its user upon determining that a delay occurs."

The present claim terminology recites "determines whether each of the communications between the client and server lasts longer than a specific time, and upon determining that the specific time has been exceeded, causes a message provided in the code to be presented to a user."

Coker et al. teaches a busy state manager configured to monitor and inform a user of the status and progress of a request [0308]. (It is noted that a request is interpreted as a communication between a client and server)

The claim terminology "determines" is interpreted as 'to find out or come to a decision about by investigation, reasoning, or calculation" (http://www.merriam-webster.com/dictionary/determine) Since the client computer monitors and informs a user of the status of the request, the client computer is understood as at least investigating whether a communication, i.e., request, will last longer than a specific time. In response to the investigating, the client informs the user about the status of the request.

[B] Applicant argues that Coker "alerts the client user about predicted, not actual, delays.....Coker's approach contrasts sharply with the present subject matter, where the message regarding delayed client-server communications is sent upon determining that the delay occurs." Applicant's claim terminology does not incorporate an actual delay.

Coker et al. implies that the server will continuously inform the user about a delay ([0309] e.g., the server will continue to inform the client of the progress of the request...upon receiving the progress

information from the server, the client can update a progress bar). Additionally, Coker et al. appears to teach that the status of this request is based on two variables. The first is that the user is informed that the request will take a long time to finish [0308]. In effect, the user is informed via a message that the request will take longer than expected. Also, the second variable would be the actual progress of the request, i.e., initiated, near completion, or completed ([0309] e.g., it is inferred that the progress bar shows the amount of completion of a request that is pending (0% completion), half way complete (50%), and complete (100%). Arguably, a progress bar shows the pending completion or delay in completion of a request.

Thus, when the client investigates, i.e., determines, that the request will take a long time, the user is informed about why their request is delayed and how long it may be delayed. Also, it may be argued that a predicted delay is an actual delay because it is known with certainty that a delay is going to occur. Upon deciding that a delay will occur, the user is informed about this delay and informed of the status of their request.

[C] Applicant argues that Coker waits a period of time before locking the client device, but that the present subject matter teaches that input is blocked during each of the communications between the client and server. Coker et al. teaches that the client is locked when the request is processed ([0309] e.g., lock the user interface during the time the request is being processed) It appears that either the client is locked upon determining a delay will occur or that when the request is actually being processed, the client is then locked. Coker et al. does not state that the initial request is terminated or stopped upon the server determining the delay, but rather, as part of the same communication with the server, this same request is processed following a determination of potential delays.

The claim language reads that the code, which when executed, is configured to block client input during a communication with the server. Coker et al. teaches that the code is executed at a point in time following a request from the client to the server [0308]. It appears that that the point in time in which the client is locked is when the request is processed following the determination that the request will take a long time to finish. Since the request is first assessed for delays and later processed as part of the same client-server communication, it is understood that the code, when executed during this communication, will lock the client interface.

Response to Amendment

1. The response under 37 CFR 1.132 filed 8/6/2009 is insufficient to overcome the rejection of claim 1 based upon Coker et al. as set forth in the last Office action.

2. The Examiner agrees with applicant's assessment of Coker et al. in paragraphs 1-6 with the exception that Coker suggests that the particular client application makes the request.

3. As per paragraph 7, applicant states that Coker et al. teaches a predictive approach which does not determine that a client request takes longer than a specific time to complete. In response, it is interpreted that in order to assess that a request is going to take longer than necessary compared to a request that is processed in a nominal time, it is necessary to reference a time from which a 'longer than necessary' time period can be deduced. The Examiner interpreted, through implication, that the time period that is referenced is the specific time period for the particular application. Moreover, with regard to the level of user inconvenience due to the server not predicting delays that are due to poor network performance, the Examiner does not see this issue addressed within the claim language. The claim language does not teach any

mechanism for taking into possibly delays that Coker et al. fails to include during client-server communication.

4. As per paragraph 8, applicant states that it appears Coker et al. does not lock the interface until after some time after the request is transmitted. The language in Coker does present ambiguity as to whether the client is locked regardless of the time the request takes to be processed or whether the client is locked if and only if the request takes longer than necessary to process. In response, the Examiner modified Coker, as addresses later in the prior Office Action, to illustrate programs such as Outlook in which a delete operation is used. During this time, the client interface is locked so as to not interfere with the delete process. Recognizing that Coker may be read as locking the interface during extended requests, the Examiner recognized that Coker does illustrate the capability of locking the interface. This code may be implemented in other user sensitive applications to prevent potential corruption.

5. As per paragraph 9, applicant discusses that the visual indicator may be distracting to the user because they do not take into account whether completion of the current request is actually delayed. Again, applicant's claim language does not teach any feature that avoids user distraction.

6. As per paragraph 10, applicant summarizes Coker and discusses the difference between an actual time vs. a predicted time. Again, applicant's claim language recites a specific time which could readily be a predicted time or an actual time. A predicted time has a numerical value and therefore is interpreted as being specific. Applicant's claim language does not define how the time is determined.

7. As per paragraph 11-13, applicant discusses Balducci synchronization mechanisms. However, the Examiner introduced Balducci to illustrate a potential application from which a locking mechanism could be utilized. The Examiner does agree with applicant's assessment of Balducci but did not see a discussion pertaining how a delete function would use implement a locking mechanism during the delete operation.

8. In summary, the Examiner is applying the locking code, as per Coker et al., to be applicable to other applications that require locking out a user during a sensitive operation, such as deleting contents from a server via a client request. In the event this communication takes longer than expected, then as a courtesy to the user, the user is informed because it is understood that a user has an initial completion expectation. If the processing should take longer than expected, then this user should be notified because the terminal will be locked during the initial request.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DARRIN DUNN whose telephone number is (571)270-1645. The examiner can normally be reached on EST:M-R(8:00-5:00) 9/5/4.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Albert DeCady can be reached on (571) 272-3819. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/DD/

03/01/10

/Albert DeCady/

Supervisory Patent Examiner

Art Unit 2121